REMARKS

Introduction

Claims 1-19 are pending, with new claims 17-19 having been added hereby. Claims 1, 7, 8, 9, 14, and 15 have been amended. The additions and amendments to the claims do not present new matter. Reconsideration of the rejection of the pending claims is respectfully requested in view of the above amendments and the following remarks.

The indefiniteness rejection should be withdrawn

Claims 1, 7, 8, 14, and 15 have been rejected under 35 U.S.C. § 112, second paragraph, as being indefinite due to informalities. Since each of these claims has been amended to remove the informalities noted in the Office Action, it is respectfully submitted that the rejection has been obviated, and that claims 1, 7, 8, 14, and 15 are definite. Withdrawal of the indefiniteness rejection is therefore respectfully requested.

The rejection of claims 1-16 under 35 U.S.C. § 103(a) should be withdrawn

Claims 1-16 have been rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 5,031,089 to Liu (hereinafter "Liu") in view of U.S. Patent No. 6,477,586 to Achenson (hereinafter "Achenson").

In the Office Action, the Examiner essentially argues that the network node load-balancing method disclosed in Liu is analogous to the method of claim 1 (which does not refer to networks or network nodes) and that the "it would have been obvious to . . . apply the ideology [of Liu] to a thread [use of threads is discussed in Achenson] and its associated queue rather than a computer node since in a case where a balanced workload is sought in a single multithreaded system rather than a multinode network the desired effect is equivalent." Applicant respectfully traverses the Examiner's contention that the threads to which call flow events are distributed according to claim 1 are analogous to the network nodes of Liu. Threads are independent execution paths for computer instructions and are not hardware "entities" in the manner of network nodes. Thus, the process of monitoring and making determinations regarding the efficiency of execution of a thread is not analogous to, and not rendered obvious by, monitoring the workload status of network nodes and performing load balancing thereon.

Moreover, Applicants emphasize that the Liu reference is a particularly inappropriate reference to base an obviousness rejection against the present invention because Liu directly teaches away from the present invention. More specifically, Liu plainly states that a server-initiated approach, i.e., a load-balancing process in which the nodes themselves are self-monitoring and initiate load-balancing procedures, is superior to a centrally managed load-balancing approach, which the present invention employs.

As an indication of the centrally managed character of the load-balancing process of the present invention, the specification provides that "[c]all flow thread manager 318 interacts with call flow engine 316 to manage the multiple threads handling call flow events within call flow server 300. Call flow thread manager 318 distributes call flow events among the respective call flow queues associated 320." (Specification page 15, lines 25-28). The specification further provides that the call flow manager allocates call flow events among the threads, and thereafter determines the activity on each thread and whether any of the threads has exceeded a maximum call flow capacity. (See Specification, page 19, lines 1-3). If a thread has exceeded the maximum call flow capacity, the call flow manager then allocates the excess call flow load to another thread. (Id. at bottom paragraph, lines 2-4). According to this centralized approach, the call flow manager monitors the threads, makes determinations as to whether load imbalances exist among the threads, and then initiates reassignment to re-balance the call flow load to address imbalances.

The teachings of Liu are completely contrary, and would clearly discourage one of skill in the art from implementing a centralized approach. As evidence of this discouragement, Liu states:

> It was determined by the inventors herein that an ideal resource-sharing algorithm should have the following characteristics.
>
> . . . .
>
> 2) Decentralized: each processing computer node should determine, on its own, whether to process a job locally or to send the job to some other node for processing. ***There is no need for a central dispatcher. Since the central dispatcher is not required, the***

problem of a single point of failure is eliminated.
(Liu, col. 8, lines 43-53; emphasis added).

As indicated in the above-quoted paragraph, Liu asserts that a centralized approach is inferior to a decentralized approach in that the centralized approach is subject to the problem of failure of the centralized manager itself, which is obviated by the decentralized approach.

Since a reference may be said to "teach away" from the claimed subject matter when a person having ordinary skill in the art is discouraged from following the direction set out in the reference or is led in a different direction from the one taken by the applicant, see In re Gurley, 27 F.3d 551 (Fed. Cir. 1994), it is respectfully submitted that Liu teaches away from the claimed subject matter where it advocates the superiority of the decentralized over a centralized scheme, and leads the skilled practitioner in a different direction from the one taken in the present invention. Therefore, there would be no motivation to combine Liu and Achenson in an attempt to arrive at the present claimed invention of claim 1. Claim 8 recites a computer program product with similar limitations as claim 1, so the same argument against using Liu and Achenson to support an obviousness conclusion with respect to claim 8 applies.

To remove any possible ambiguity, the preamble of independent claim 1 has been amended to recite a method that is performed at a manager, and the preamble of independent claim 8 has been amended to recite a program code that is operable at a manager. It is therefore submitted that independent claims 1 and 8, and their respective dependent claims 2-7 and 9-14, are patentable over Liu and Achenson.

Independent claim 15 recites an apparatus for distributing call flow events among a plurality of threads that includes a call flow engine configured to execute call flow events associated with one of the threads and a call flow manager configured to distribute a plurality of call flow events among a plurality of threads used for managing the processing of a plurality of call flows. As neither Liu nor Achenson discloses or suggests an apparatus including a call flow manager configured to to distribute a plurality of call flow events among a plurality of threads, independent claim 15 and its dependent claim 16 are also patentable over these references.

In light of the foregoing, withdrawal of the obviousness rejection of claims 1-16 based upon Liu and Achenson is requested.

New Claims 17-19 are also patentable

New claim 17 depends from claim 1, new claim 18 depends from 8, and new claim 19 depends from claim 15. Thus, new claims are patentable at least for the same reasons given above with respect to allowable independent base claims 1, 8 and 15.

Conclusion

All issues having been addressed, it is believed that the present application is in condition for allowance. Prompt reconsideration and allowance of the present application are respectfully requested. Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned "Version Of Amendment Showing Changes Made."

Respectfully submitted,

KENYON & KENYON

Date: May 12, 2003

[illegible]

Jong H. Lee
Registration No. 36,197

KENYON & KENYON
One Broadway
New York, NY 10004

CUSTOMER NO. 26646

VERSION OF AMENDMENT SHOWING CHANGES MADE

Please amend claims 1, 7, 8, 9, 14, and 15 as follows:

1. (Amended) In a computer system, a method, performed at a manager, of distributing call flow events among a plurality of threads, each thread having an associated call flow event queue in which call flow events are queued, the method comprising:

A. determining a call flow workload level for each of the plurality of threads;
B. determining that a first of the plurality of threads is inefficiently handling its assigned call flow workload; and
C. reassigning a call flow event from the call flow event queue associated with the first thread to the call flow event queue associated with a second of the plurality of threads.

7. (Amended) The method according to claim 1, wherein step A comprises:

A.1 assigning call flow events among the call flow queues associated with the respective plurality of threads in the system.

8. (Amended) A computer program product for use with a computer system, the computer system operatively coupled to a computer network and capable of communicating with one or more processes over the network, the computer program product comprising a computer usable medium having program code embodied in the medium, the program code being operable at a manager and comprising:

(A) program code configured to determine a call flow workload level for each of the plurality of threads;
(B) program code configured to determine that a first of the plurality of threads is inefficiently handling its assigned call flow workload; and
(C) program code configured to reassign a call flow event from the call flow event queue associated with the first thread to the call flow event queue associated with a second of the plurality of threads.

9. (Amended) The computer program product of [ ] claim 8, further comprising [program]:

(D) program code configured to process the call flow events within each of the plurality of threads.

14. (Amended) The computer program product according to claim 8, further comprising:

(A.1) program code configured to assign call flow events among the call flow event queues associated <u>with</u> the respective plurality of threads in the system.

15. (Amended) In a computer system, an apparatus for distributing call flow events among a plurality of threads, each thread having an associated call flow event queue in which call flow events <u>are</u> queued, the apparatus comprising:

a call flow engine configured <u>to</u> execute call flow events associated with one of the threads;

a call flow manager configured to distribute a plurality of call flow events among a plurality of threads used for managing the processing of <u>a</u> plurality of call flows, [n] the call flow manager optimizing the processing of the call flows by determining which <u>of the</u> plurality of threads are operating inefficiently and reassigning a portion of the call flow events assigned to the inefficient thread to other of the plurality of threads having excess call flow processing capacity.